Kotlin language specification

Version 1.9-rfc+0.1

Marat Akhin Mikhail Belyaev

Chapter 15

Runtime type information

The *runtime type information* (RTTI) is the information about Kotlin types of values available from these values at runtime. RTTI affects the semantics of certain expressions, changing their evaluation depending on the amount of RTTI available for particular values, implementation, and platform:

- The type checking operator
- The cast expression, especially the `as?` operator
- Class literals and the values they evaluate to

Runtime types are particular instances of RTTI for a particular value at runtime. These model a subset of the Kotlin type system. Namely, the runtime types are limited to classifier types, function types and a special case of `kotlin.Nothing?` which is the type of `null` reference and the only nullable runtime type. This includes the classifier types created by anonymous object literals. There is a slight distinction between a Kotlin type system type and its runtime counterpart:

- On some platforms, some particular types may have the same runtime type representation. This means that checking or casting values of these types works the same way as if they were the same type
- Generic types with the same classifier are not required to have different runtime representations. One cannot generally rely on them having the same representation outside of a particular platform. Platform specifications must clarify whether some or all types on these platforms have this feature.

RTTI is also the source of information for platform-specific *reflection* facilities in the standard library.

The types actual values may have are limited to class and object types and function types as well as `kotlin.Nothing?` for the `null` reference. `kotlin.Nothing` (not to be confused with its nullable variant `kotlin.Nothing?`) is special in the way that this type is never encountered as a runtime type even though it may

have a platform-specific representation. The reason for this is that this type is used to signify non-existent values.

15.1 Runtime-available types

Runtime-available types are the types that can be guaranteed (during compilation) to have a concrete *runtime* counterpart. These include all the runtime types, their nullable variants as well as `reified` type parameters, that are guaranteed to inline to a runtime type during type parameter substitution. Only runtime-available types may be passed (implicitly or explicitly) as substitutions to reified type parameters, used for type checks and safe casts. During these operations, the nullability of the type is checked using reference-equality to `null`, while the rest is performed by accessing the runtime type of a value and comparing it to the supplied runtime-available type.

For all generic types that are not expected to have RTTI for their generic arguments, only "raw" variants of generic types (denoted in code using the star-projected type notation or a special parameter-less notation) are runtime-available.

> Note: one may say that classifier generics are *partially* runtime available due to them having information about only the classifier part of the type

Exception types must be runtime-available to enable type checks that the `catch` clause of `try`-expression performs.

Only non-nullable runtime types may be used in `class` literal expressions. These include reified type parameters with non-nullable upper bounds, as well as all classifier and function types.

15.2 Reflection

Particular platforms may provide more complex facilities for runtime type introspection through the means of *reflection* — special platform-provided part of the standard library that allows to access more detailed information about types and declarations at runtime. It is, however, platform-specific and one must refer to particular platform documentation for details.